SPECIFICATION

Docket No. 0544MH-40063

TO ALL WHOM IT MAY CONCERN:

BE IT KNOWN that we, Craig Dunn and Conor McGann, residing in the State of Texas, have invented new and useful improvements in a

MESSAGING SYSTEM FOR COMPUTERS

of which the following is a specification:

1

CROSS REFERENCE TO RELATED APPLICATION

This Application claims the benefit of U.S. Provisional Application No. 60/187,342 filed March 6, 2000.

BACKGROUND OF THE INVENTION

1. Field of the Invention:

The present invention relates generally to computer systems, and more specifically to message handling between processes and a distributed computer system.

2. Description of the Prior Art:

As electronic computer systems and the processes executed on the systems become more complex, it is common to distribute processing in some manner. Further, as computing becomes more distributed over communication networks, including worldwide networks such as the internet, distributed processing becomes a desirable approach to computing.

When processing on problems becomes distributed, whether locally or remotely, communication between various processes becomes more important. These processes are sometimes referred to as modules, and must communicate with each other to perform the overall processing tasks. One common technique for communication between remotely located processes is the use of message

passed between them. By using messages, it is not necessary that the processes be tightly coupled, and each process can be optimized to perform its function.

Available systems and methods for passing messages between distributed processes are not reliable enough for some applications. In certain critical applications, failure of a software or hardware module could interfere with transfer of a message between processes. If a message is lost or garbled, it may be difficult and impossible for the processes involved to accurately recover from the failure. Also, many such systems are not fast enough to ensure adequate response times as seen by users.

It would be desirable for an improved messaging system and method to reliably handle messages, and insure that messages between processes are properly routed and confirmed as received by the intended recipient process.

SUMMARY OF THE INVENTION

In accordance with the present invention a messaging system utilizes a local queue manager to receive messages intended for other processes. Messages received by the queue manager are stored onto a local persistent storage device, and a process sending the message has completed the sending action. The local queue manager then sends the message to an appropriate recipient. When the message has been received and confirmed, the recipient removes the message from the persistent storage device. If a hardware or software failure occurs, the message is stored and can be re-sent after the failure is corrected.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself however, as well as a preferred mode of use, further objects and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 and 2 are block diagrams illustrating a messaging system in accordance with the present invention; and

Figure 3 is a block diagram illustrating handling of multiple messages between processes.

DESCRIPTION OF THE PREFERRED EMBODIMENT

As will be appreciated by those skilled in the art, the system described below can be implemented on nearly any system that passes messages between processes. Preferably, messages are handled asynchronously. This means that once the message is sent from the sending process, timing of receipt of the message by the receiving processes is generally not critical. In the description below, it is assumed that the sending and receiving processes are not tightly coupled, and that a message once sent need not have delivery confirmed.

Referring to Figure 1, a sending process 20 needs to send a message, as generally known in the art, to receiving process 22. This message can be, for example, a remote database access query, a status request, notification of an event occurring within sending process 20, or any other subject matter suitable for such remote messaging.

The message is sent from sending process 20 to message handling system 24, which eventually routes the message to receiving process 22. The system and method described herein are particularly applicable to the design and operation of the message handling system 24.

Referring to Figure 2, message 26 is sent to messaging collector 28. Messaging collector 28 serves as the interface to the outside world for message handling system 24. Messaging handling collector 28 is preferably an object that is invoked by sending process 20 in order to handle the message to be sent. In a preferred embodiment, messaging collector 28 is invoked by a Java class call, and

invokes a single method for each method passed. Those skilled in the art will recognize that messaging collector 28 may be implemented differently to perform the same functions as described herein.

Messaging collector 28 immediately passes the message off to local queue manager 30. Because local queue manager 30 is local to sending process 20, this happens very quickly and results in minimal latencies. Once messaging collector 28 has passed the message off to local queue manager 30, sending process 20 has completed sending message 26, and may return to other functions.

This approach causes message handling to be asynchronous as possible. From this point forward, the message handling system 24 will handle the responsibility for delivering message 26.

The messaging collector method 28 preferably returns a status code to sending process 20 indicating success or failure for accepting the message. This is based upon a similar success or failure status received from the local queue manager. Once an indication of success is returned, the sending process 20 assumes that the message will be reliably delivered

Local queue manager 30 accepts messages as quickly as possible from messaging collector 28, and prepares them for continued transmission. Preferably, multiple messaging collector objects 28 communicate with local queue manager 30. Thus, local queue manager 30 can receive numerous messages from different messaging collector methods 28, which are in turn evoked by one or more sending processes. When the message is received by the main thread of local queue

manager 30, the message is queued to a static internal class vector. Also, preferably, it is persisted to a local file system 32, where it is stored until the message is delivered. Until removed by the receiver, the message remains on the local file system so that it can be retrieved and resent in case of a hardware or software failure somewhere along the line. This increased tolerance of hardware and network failures occurs at a cost of slightly increased overall latency. However, for applications that need to ensure message delivery, this tradeoff is a good one.

Local queue manager 30 also includes a separate thread to manage outbound traffic. This thread pulls messages off the internal queue, preferably in FIFO order, and makes a call to a configured routing object 34. The call includes at least a message destination taken from the message itself, and router 34 returns a destination server to local queue manager 30. Once the destination server has been obtained, the message is passed by a standard messaging system to such server.

The destination server identified by router 34 can be implemented in any identifying manner compatible with the remainder messaging system. As described below, the message is passed to a messaging writer 36, the identification and location of which is stored in a registry or similar routing table. The routing algorithms utilized by router 34 can be any that are appropriate to the specific implementation, and can be easily changed at any time due to the separation between router 34 and local queue manager 30. This allows the routing algorithm to be changed or enhanced when, for example, the overall system is expanded and the messaging load increases.

Message writer 36 is a module that receives messages and routes them to a specific associated process. In the example in Figure 2, plug in processes 38, 40 are associated with message writer 36, and receive messages from it.

Within messaging writer 36, a primary thread preferably receives incoming messages, and places them on to a background thread queue. The background thread determines the type of message and routes it to a specific queue for the associated plug in 38, 40. These queue threads then simply pass the messages to the plug ins 38, 40 for final receipt and process.

Once the message writer 36 has delivered the message, it removes the message form the local queue, where it had been placed by the local queue manager 30. Until message writer 36 has delivered the message, it remains on the persistent storage device, from where it can be accessed later if need be. Thus, in the event of a failure, the message is saved in a reliable location. Once the failure has been corrected, the message can be resent and receipt insured by plug in 38, 40, the intended recipient. This persistent storage of messages on a local file server enables recovery from failures to be performed in a simple manner, without requiring the messages to be re-generated by sending processes 20, a process which may not be possible due to the nature of the sending process.

The latency of messages handled by such system can be controlled to be acceptably small. Initially, the local latency from sending, or client, processes 20, are very small, due to the local nature of messaging collector 28 and local queue manager 30. End-to-end latency, defined as the time it takes a process to send a

message to plug-in 38 or 40, depends on the detailed design of message handling system 24 and the current load the system is handling. Message handling system 24 can be increased in size if a lower end-to-end latency is required, and still function in the same manner as described above.

Referring to Figure 3, in block diagram of a more complete message handling system 24 is illustrated. Multiple messages 51-56 are generated by multiple sending processes (not shown). Each message 51-56 is sent to its associated messaging collector 61-66. In this example, three local queue managers 71, 72, 73 are used, and each message is sent to an associated local queue manager as shown. Each local queue manager then persists, or stores in a long-term manner, each message to a local file system 74, 75, 76. Each local queue manager 71-73 invokes an associated router method 81, 82, 83, which handles routing of messages to appropriate messaging writers 91, 92, 93.

Message writers 91-93 determine the underlying type of message, and route them to an appropriate plug in as shown. As described previously, the appropriate plug in converts the message to the appropriate format, and forwards its to the receiving process 22.

As can be seen from Figure 3, message handling system 24 is extremely fast, and can be easily extended. The message handling system is not concerned with the type of underlying message, but transfers all messages in the same way. The underlying type of the message is only a concern when the message reaches the appropriate messaging router, which sends it to the appropriate plug in for

reconversion back to the original message type. All message types are treated in the same manner by the message transport system.

Additional local queue managers and messaging writers can be added at any time. As described above, routing algorithms are easily modified at any stage to perform routing that is either optimal, or good enough for the specific implementation.

The system described above is simple in operation and economical in design. It allows messages to be reliably transmitted and received, even in the event of a system failure, because the message is not marked as received by the local queue manager until it is stored onto a reliable long term storage system. Because the message remains in persistent storage until removed after receipt, it can always be re-sent after a failure.

Because the transport system does not care about the type of the underlying message, it can be used for messages of any type. The message collector 28 formats any type of message into a standard type, and it is returned to its original type in the appropriate plug in selected by the message writer 36. Thus, multiple new message types can be added to the system without impacting the message handling performed on the messages. Whenever a new message type is defined, the message collector method 28 is updated, and a new plug in is prepared to convert those messages back into their original type. No additional modifications are required.

While the invention has been particularly shown and described with reference to a preferred embodiment, it will be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention.